Assginment1 :
Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

Ans:
All Colums: Select * from customers;
Only Coustomer Name and Email:
select customer_name, email_address from customers
where city = 'Specific City';
==========
Assignment2:
Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

Ans:
Inner Join:
```
select c.customer_name, c.region, o.order_date, o.total_amount
from customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
where c.region = 'Specified Region';
```
Left Join:
```
select c.customer_name, c.region, o.order_date, o.total_amount
from customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
where c.region = 'Specified Region';
```
==========
Assignment3:
Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

Ans:
Find Coustomers:
```
SELECT c.customer_name, o.order_date, o.total_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.total_amount > (
  SELECT AVG(total_amount)
  FROM orders
);
```

Union Query:
```
SELECT 'Premium Customer' AS customer_type, c.customer_name, o.order_date, o.total_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.total_amount > (
  SELECT AVG(total_amount)
  FROM orders
)

UNION

SELECT 'Regular Customer' AS customer_type, c.customer_name, o.order_date, o.total_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
```

```
WHERE o.total_amount <= (
  SELECT AVG(total_amount)
  FROM orders
);
```

==========

Assignment4:

Compose SQL statements to BEGIN a transaction, INSERT a new record into the ’orders’ table, COMMIT the transaction, then UPDATE the ’products’ table, and ROLLBACK the transaction.

Ans:

-->begin

```
BEGIN TRANSACTION;
```

-->insert

```
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (1, ’2022-01-01’, 100.00);
```

-->commit

```
COMMIT;
```

-->update

```
UPDATE products
SET quantity_in_stock = quantity_in_stock - 1
WHERE product_id = 1;
```

-->rollback

```
ROLLBACK;
```

==========

Assignment5:

Begin a transaction, perform a series of INSERTs into ’orders’, setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

Ans:

-->Begin

```
BEGIN TRANSACTION;
```

-->Insert

```
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (1, ’2022-01-01’, 100.00);
```

-->savepoint

```
SAVEPOINT sp1;
```

-->Insert

```
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (2, ’2022-01-05’, 200.00);
```

--> savepoint

```
SAVEPOINT sp2;
```

--> Insertr

```
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (3, ’2022-01-10’, 300.00);
```

```
-- Roll back to the second savepoint
ROLLBACK TO SAVEPOINT sp2;
```

```
-- Commit the overall transaction
```

COMMIT;
==========
Assignment6:
Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Ans:

Transaction logs are a crucial component of database management systems, playing a vital role in ensuring data integrity and facilitating data recovery in the event of unexpected failures or shutdowns. A transaction log is a record of all changes made to a database, including inserts, updates, and deletes, which allows the database to maintain a consistent state even in the face of failures.

The transaction log serves as a safety net, enabling the database to recover to a consistent state by replaying the log entries in the event of a failure. This ensures that all committed transactions are durable and that the database remains in a consistent state.

Hypothetical Scenario:

Company: Online Banking System (OBS) Database: MySQL Scenario: Unexpected Server Shutdown

On a busy Friday afternoon, the OBS server suddenly shuts down due to a power failure, causing the database to become unavailable. The database administrator, Rachel, is notified and quickly springs into action to recover the database.

Pre-Crash State:

Before the shutdown, several transactions were in progress, including:

---A customer, John, initiated a transfer of $1,000 from his checking account to his savings account.
---Another customer, Emily, was in the process of paying a bill of $500 using her credit card.
Recovery Process:

Rachel follows the standard recovery procedure:

---She restarts the server and initiates the database recovery process.
---The database management system (DBMS) analyzes the transaction log to determine the last consistent state of the database.
---The DBMS replays the transaction log entries, applying all committed transactions to the database.
---The DBMS rolls back any uncommitted transactions, ensuring that the database remains in a consistent state.

Recovery Outcome:

After replaying the transaction log, the database is recovered to a consistent state. The transactions that were in progress at the time of the shutdown are either committed or rolled back:

---John’s transfer of $1,000 is committed, and his accounts are updated accordingly.
---Emily’s bill payment of $500 is rolled back, as it was not committed at the time of the shutdown. Her credit card is not charged, and her account remains unchanged.
Thanks to the transaction log, the OBS database is recovered to a consistent state, ensuring that customer data is accurate and up-to-date. The quick recovery minimizes downtime, and online banking services are restored, allowing customers to access their accounts and conduct transactions without interruption.